Scraping de páginas dinâmicas e solução de caminho com grafos

Alan da Mota Nascimento[1], **Gabriel Alberto Moura de Sá**[1]

[1] Instituto de Educação Superior de Brasília (IESB)

gabrielsa08@gmail.com, alan.nascimento@iesb.edu.br

Resumo. *O uso de Web Scrapers é uma famosa alternativa para extrair dados de forma automática de websites. Ao programar um, uma das primeiras etapas é descobrir se o site a ser "garimpado" é estático ou dinamico. Websites estáticos podem ser processados rapidamente com uma requisição, já websites dinâmicos precisam esperar pela execução de seu JavaScript antes de se obter o HTML final, impondo uma etapa a mais. Para explorar o tempo gasto no Scraping de páginas dinamicas, neste trabalho será desenvolvido uma plataforma que, utilizando grafos, consiga traçar a rota com o menor custo dado um conjunto de destinos fornecidos pelo usuário. Os valores que alimentarão este grafo ponderado serão obtidos através dos preços de passagem provenientes do Scraping de uma página dinâmica.*

1. Introdução

Atualmente, ao criar um pacote de viagens, isto é, definir diversos destinos que serão visitados de avião, é necessário pesquisar o preço das passagens individualmente até chegar a um resultado satisfatório. Este problema pode ser solucionado de forma totalmente automatica utilizando um web scrapper e algoritmos que encontram o caminho com menor custo.

Para chegar numa solução viável, certas proplemáticas devem ser investigadas. Para começar, o uso de um scrapper não é a solução ideal, posto que será necessário diversas consultas para obter os dados necessários. Levando este problema em consideração, o uso de uma API capaz de retornar os valores das passagens desejadas seria mais eficiente. Mesmo assim, será feito um scrapper para realizar o garimpo dos dados, já que não existem opções gratuitas de uma API que faça isto, reduzindo sua viabilidade.

Dado o exposto, neste artigo será exposto os obstáculos na implementação do scrapper e o problema de menor caminho, buscando encontrar uma solução viável levando em consideração o tempo gasto em cada consulta.

2. Objetivos Gerais

O objetivo neste artigo é obter uma solução viável para o Scraping de múltiplas páginas Web dinâmicas, tendo como principal métrica o tempo gasto do momento em que o usuário define o pacote de viagens até o final do cálculo do caminho de custo minimo.

3. Metodologia

3.1. Escolhendo o site para fazer o Scraping

O site escolhido para fazer os scraps foi o "https://www.decolar.com.br". Para chegar no site escolhido, alguns fatores foram levados em consideração. O site é confiável e

retorna diversos preços de passagens para cada consulta, também tem links estáticos e é estruturado de forma que os dados importantes são coletados facilmente. Além disso, mesmo em testes extensivos não houveram problemas com a quantidade de requisições sendo feitas por um único IP.

3.2. Definição do problema

Partindo do contexto de que será montado um grafo ponderado, a primeira etapa é a definição dos vértices. Para fazer isso, será solicitado ao usuário que informe destinos distintos que formarão um pacote, estes destinos vão ser os vértices do grafo. Levando em consideração que o preço das passagens variam de acordo com a data, também é necessário saber qual será a data de partida, ou seja, em que dia o usuário pretende partir do destino inicial (Chamaremos de origem) e qual o tempo em dias que ele pretende ficar em cada destino.

Utilizando os dados informados acima, é possível definir um grafo contendo N vértices, e sabendo que todos os vértices são adjacentes, é formado um grafo completo, isto é, um grafo simples em que todos os vértices possuem uma aresta conectando-os.

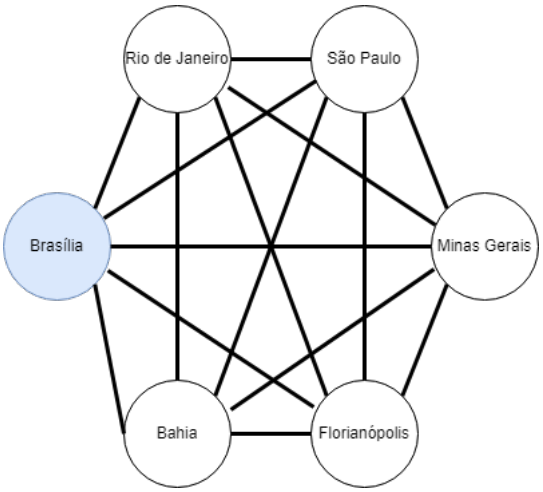

Figure 1. Definição do grafo completo contendo cinco destinos e uma origem, destacada com a cor azul.

Após definir o grafo, resta encontrar os valores de cada aresta. Esta etapa envolve o scrapping dos preços das passagens. É inviável realizar todas as consultas de uma vez, posto que estes dados são coletados numa página dinâmica e os valores das arestas variam de acordo com a data. Levando em consideração um grafo contendo N vértices e sabendo que a quantidade de datas será igual a N, para conseguir todos os valores possíveis de cada aresta seriam necessários $\frac{N(N-1)}{2} * N$ consultas, isto é, a quantidade de arestas multiplicada pela quantidade de datas possíveis.

3.3. Problema do Caixeiro Viajante

Ao definir o problema, torna-se evidente que este se enquadra no problema do caixeiro viajante. Em síntese, o usuário deverá receber qual vai ser o caminho que o permitirá visitar todos os destinos uma única vez e voltar a origem com o menor preço. Como foi explicado na seção anterior, deve ser considerado que o custo das arestas depende do

tempo, isto é, o custo de transição do vértice i até o j depende da data que o vértice i foi visitado [Boulic et al. 2008].

Com isto, é necessário trazer algumas limitações para o usuário. Como o problema do caixeiro viajante é um problema de complexidade NP-Díficil, será limitado a quantidade de destinos para seis incluindo a origem, evitando que sejam feitas consultas que tomarão muito tempo. Após isso, é ncessário encontrar um algoritmo que encontre o menor caminho com o mínimo de consultas possíveis.

4. Algoritmos

4.1. Dijkstra

A primeira solução cogitada para o problema em questão foi a utilização do algoritmo de Dijkstra, que serve para encontrar o caminho de menor custo entre os vértices de um grafo. Para isso, o algoritmo faz o cálculo do custo de deslocamento entre cada vértice de um determinado grafo, montando a partir dele um caminho que representa o menor custo.

Apesar de ser uma solução eficiente para descobrir o caminho mais efetivo entre vértices de um grafo, o algoritmo não poderia ser usado para resolver o problema citado acima, já que ele não garante que todos os vértices do grafo farão parte da solução menos custosa, podendo excluir destinos solicitados pelo usuário. Tendo isso em vista, não será utilizado o algoritmo Dijkstra neste artigo.

4.2. Vizinho mais próximo

A solução que será utilizada na aplicação é o algoritmo Vizinho mais próximo. Este é um algoritmo guloso que realizará o trajeto de trás para frente, ou seja, partirá do destino final e consultará o preço das passagens dos outros destinos para o destino final em busca do menor. Ao encontrar o destino com menor preço, este será utilizado como destino final na próxima consulta, e o processo se repetirá até que não sobre mais destinos.

Esta solução reduz a quantidade de consultas necessária por fazer decisões em cada estágio, excluindo a necessidade de consultar todos os preços da passagem ao inicializar. Porem, por ser um algoritmo guloso, há a possibilidade que o caminho escolhido não seja de fato o com menor preço, tal desvantagem é parcialmente controlada pela quantidade limitada de vértices totais, porem não seria viável utiliza-lo em maiores escalas.

Para otimizar o tempo gasto pelo scraper nas consultas, há também a alternativa de realizar as consultas concorrentemente. Para isso, em cada etapa do Vizinho mais próximo são criados N instâncias diferentes de um driver para fazer o scraping, onde N é a quantidade de destinos, e estes garimpam os preços das passagens simultaneamente.

5. Resultados obtidos

Para a tabela abaixo, foram realizadas 20 consultas com 3, 4, 5 e 6 destinos, e então calculado a média de tempo gasto em cada.

	Total destinos			
	3	4	5	6
Vizinho mais próximo	38.32s	68.88s	103.89s	147.98s
Vizinho mais próximo [Concorrente]	46.44s	69.22s	101.37s	129.02s

As otimizações que resultaram num tempo menor foram resultados de mudanças no scrapper, provando ser o principal gargalo da aplicação. Mesmo assim, realizar as consultas de forma concorrente não trouxe melhoras significativas no tempo gasto com cinco e seis destinos, e demonstrou-se mais lento com três e quatro destinos. Tal comportamento pode ser explicado pela quantidade de drivers instanciados em cada etapa do algoritmo, visto que na solução inicial apenas um é instanciado e responsável por fazer todas as consultas, e na solução concorrente devem ser instanciados N drivers para cada destino, custando mais tempo que o tempo ganho em fazer as consultas simultaneamente.

Dado o exposto acima, o algoritmo vizinho mais próximo é viável para encontrar o menor caminho, já que o scraper é a principal razão da demora para se obter o resultado e o Vizinho mais próximo é um algoritmo guloso. Também é possível reduzir o tempo do Vizinho mais próximo concorrente com um código otimizado, que não precise instanciar drivers novamente em cada etapa.

6. References

References

Boulic, R., Gamache, M., and Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. In Megiddo, N., editor, *Discrete Optimization*.